FIG. 1

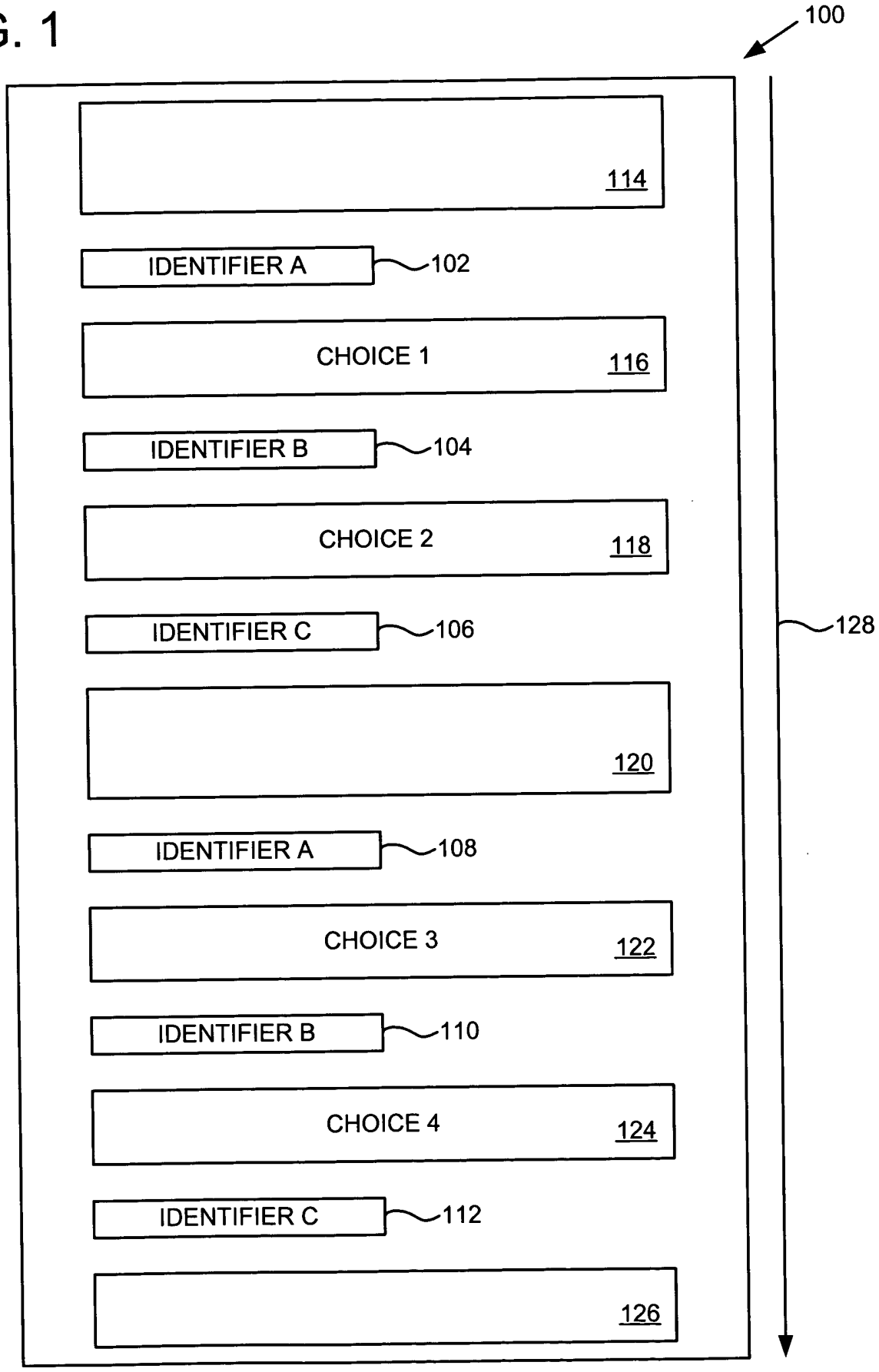

100
114
IDENTIFIER A
102
CHOICE 1
116
IDENTIFIER B
104
CHOICE 2
118
IDENTIFIER C
106
128
120
IDENTIFIER A
108
CHOICE 3
122
IDENTIFIER B
110
CHOICE 4
124
IDENTIFIER C
112
126

FIG. 2

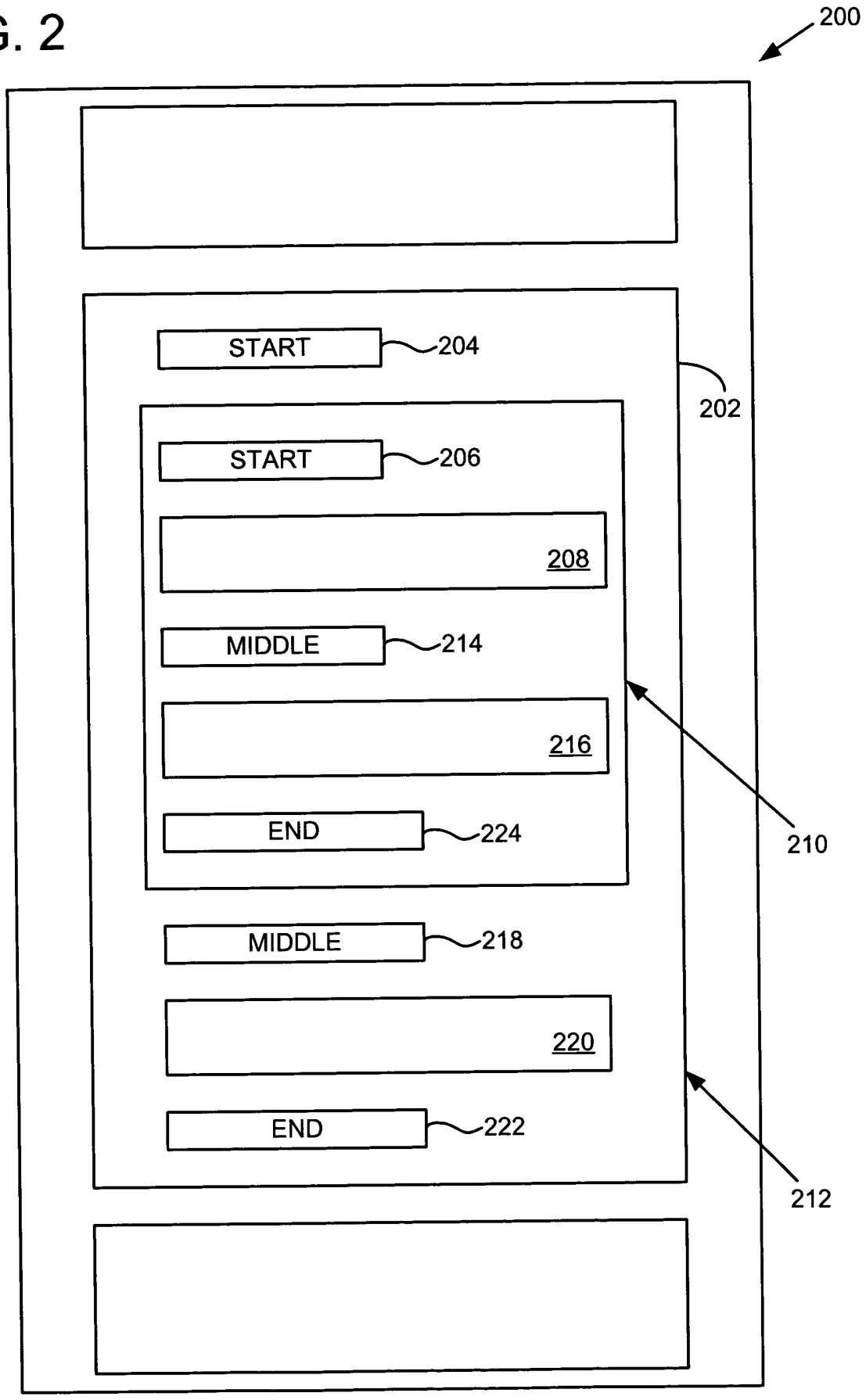

200
START
204
202
START
206
208
MIDDLE
214
216
END
224
210
MIDDLE
218
220
END
222
212

FIG. 3

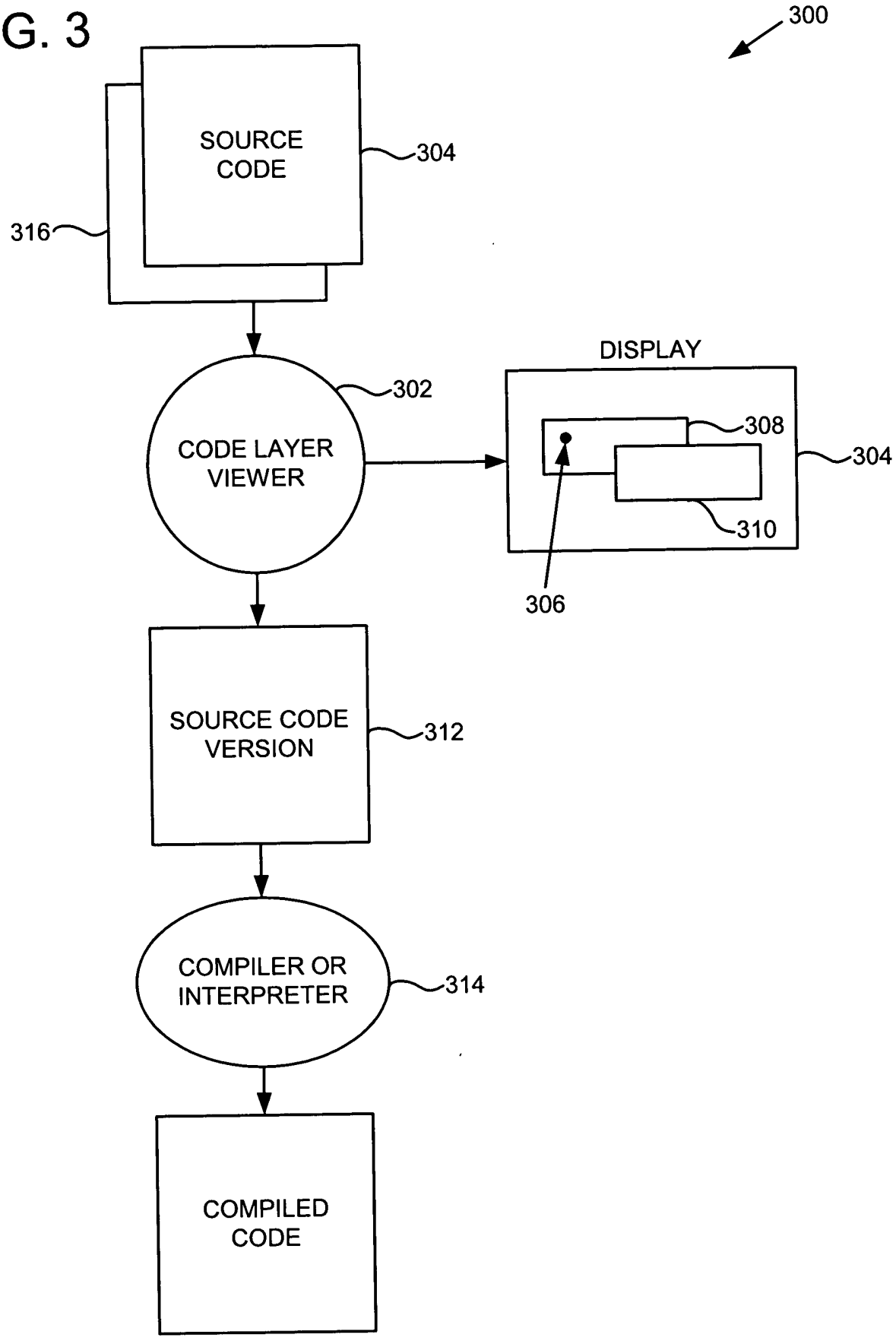

300
SOURCE CODE
304
316
DISPLAY
302
CODE LAYER VIEWER
308
304
310
306
SOURCE CODE VERSION
312
COMPILER OR INTERPRETER
314
COMPILED CODE

FIG. 4

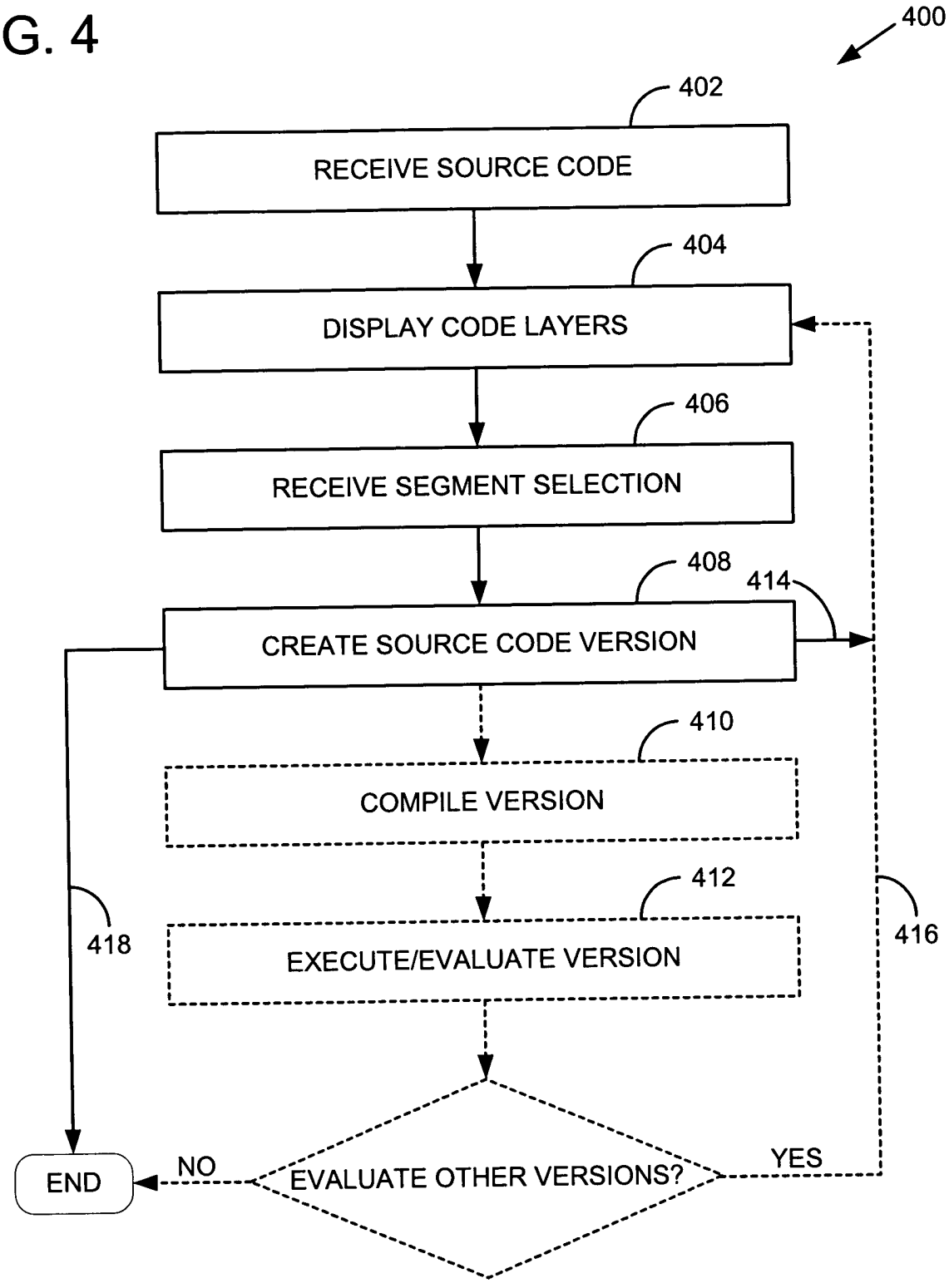

400
402
RECEIVE SOURCE CODE
404
DISPLAY CODE LAYERS
406
RECEIVE SEGMENT SELECTION
408
414
CREATE SOURCE CODE VERSION
410
COMPILE VERSION
412
EXECUTE/EVALUATE VERSION
418
416
END
NO
EVALUATE OTHER VERSIONS?
YES

FIG. 5

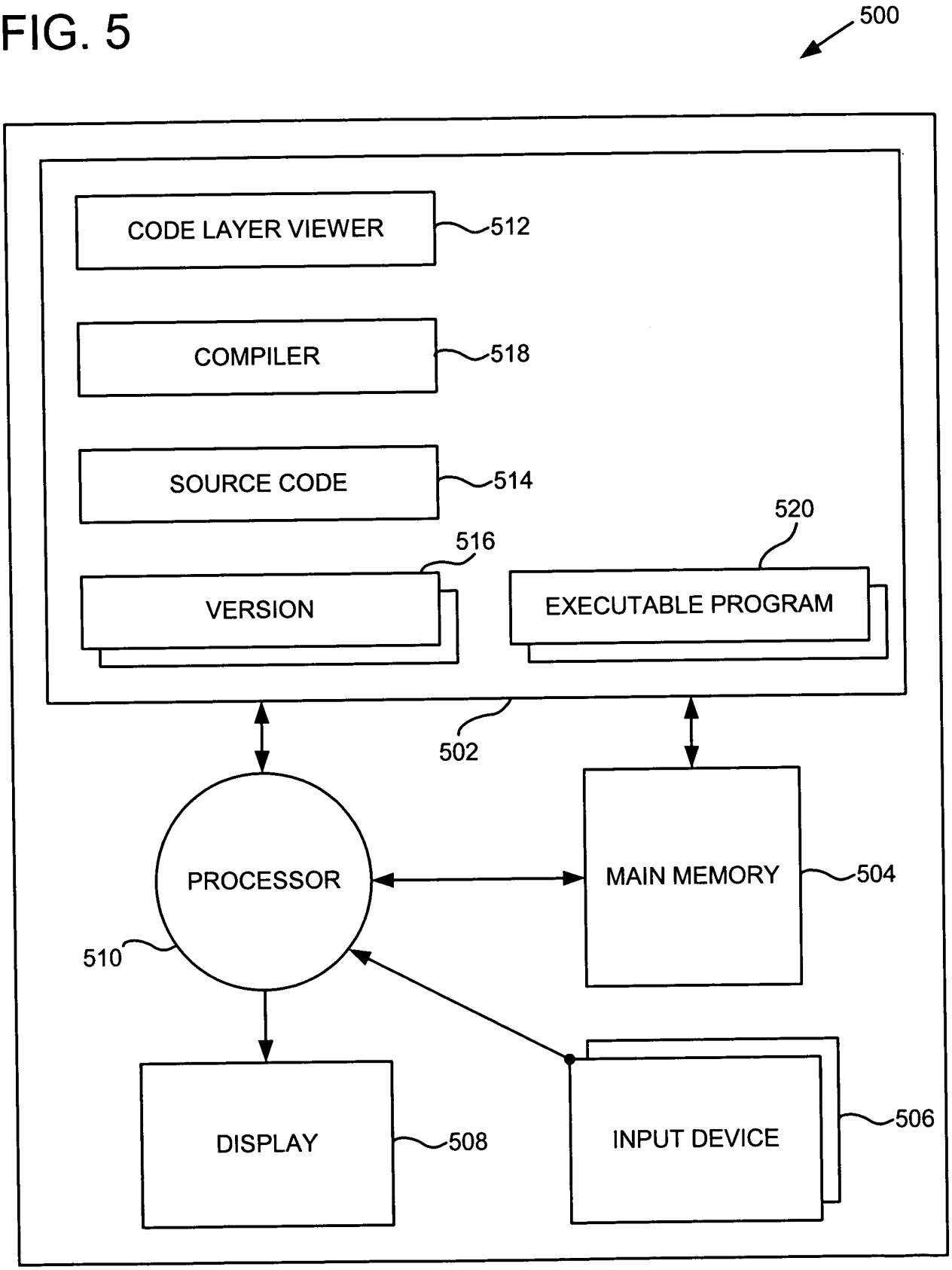

500
CODE LAYER VIEWER
512
COMPILER
518
SOURCE CODE
514
516
VERSION
520
EXECUTABLE PROGRAM
502
PROCESSOR
510
MAIN MEMORY
504
DISPLAY
508
INPUT DEVICE
506

FIG. 6

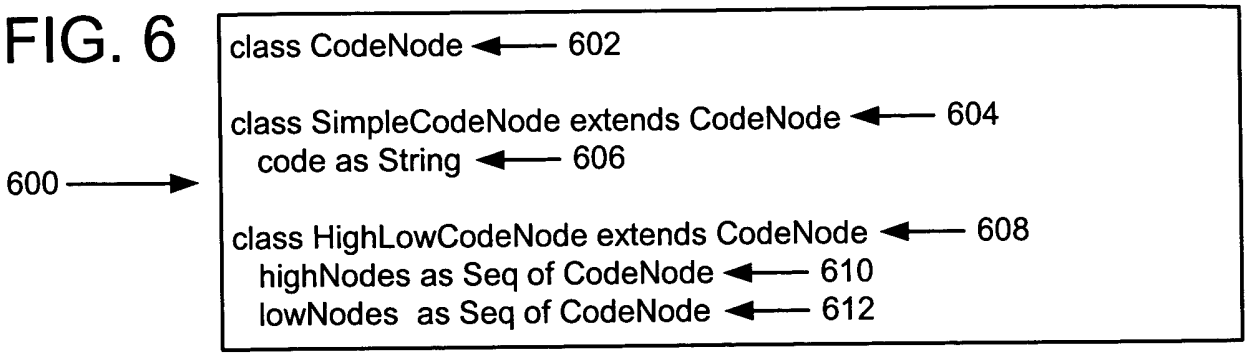


FIG. 7

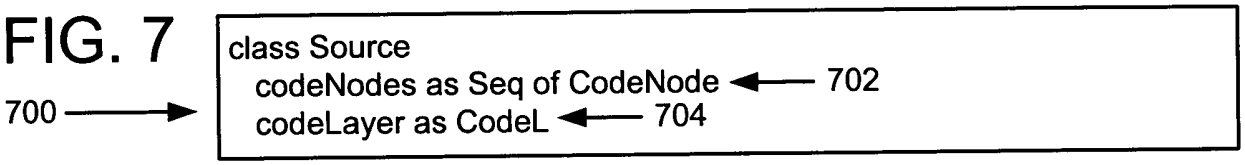


FIG. 8

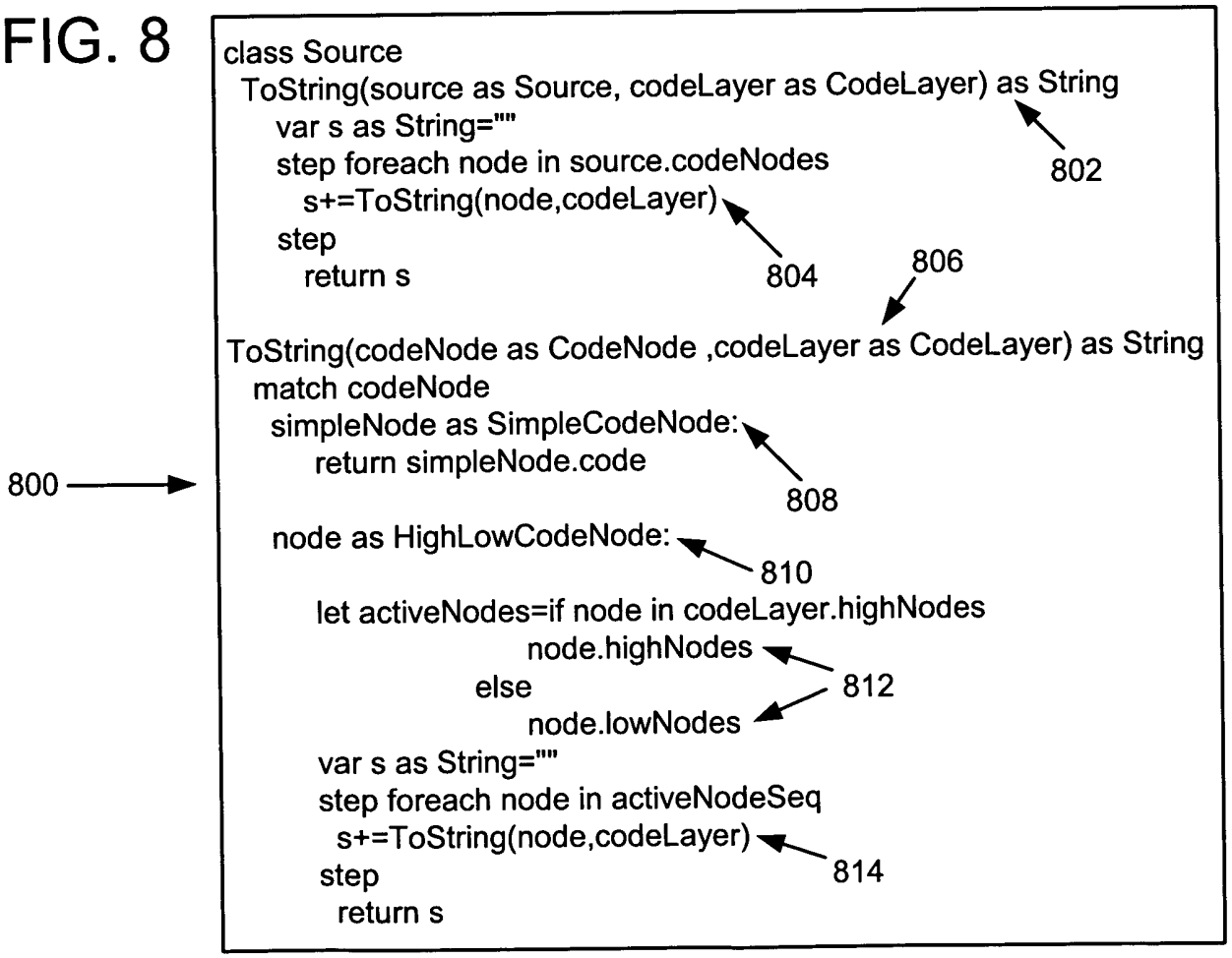


FIG. 9

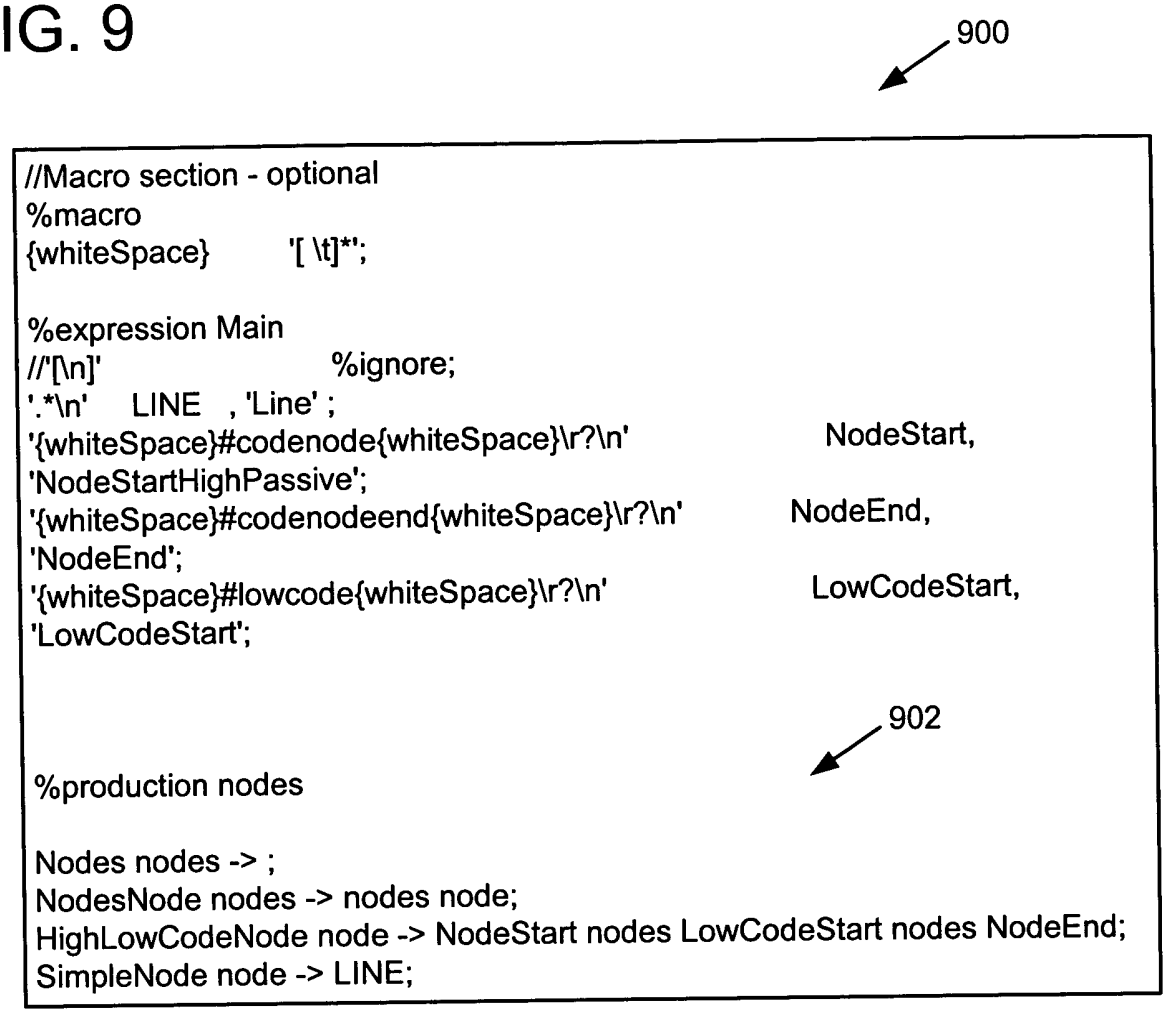

900
//Macro section - optional
%macro
{whiteSpace} '[\t]*';
%expression Main
//'[\n]' %ignore;
'.*\n' LINE , 'Line' ;
'{whiteSpace}#codenode{whiteSpace}\r?\n' NodeStart,
'NodeStartHighPassive';
'{whiteSpace}#codenodeend{whiteSpace}\r?\n' NodeEnd,
'NodeEnd';
'{whiteSpace}#lowcode{whiteSpace}\r?\n' LowCodeStart,
'LowCodeStart';
902
%production nodes
Nodes nodes -> ;
NodesNode nodes -> nodes node;
HighLowCodeNode node -> NodeStart nodes LowCodeStart nodes NodeEnd;
SimpleNode node -> LINE;

FIG. 10

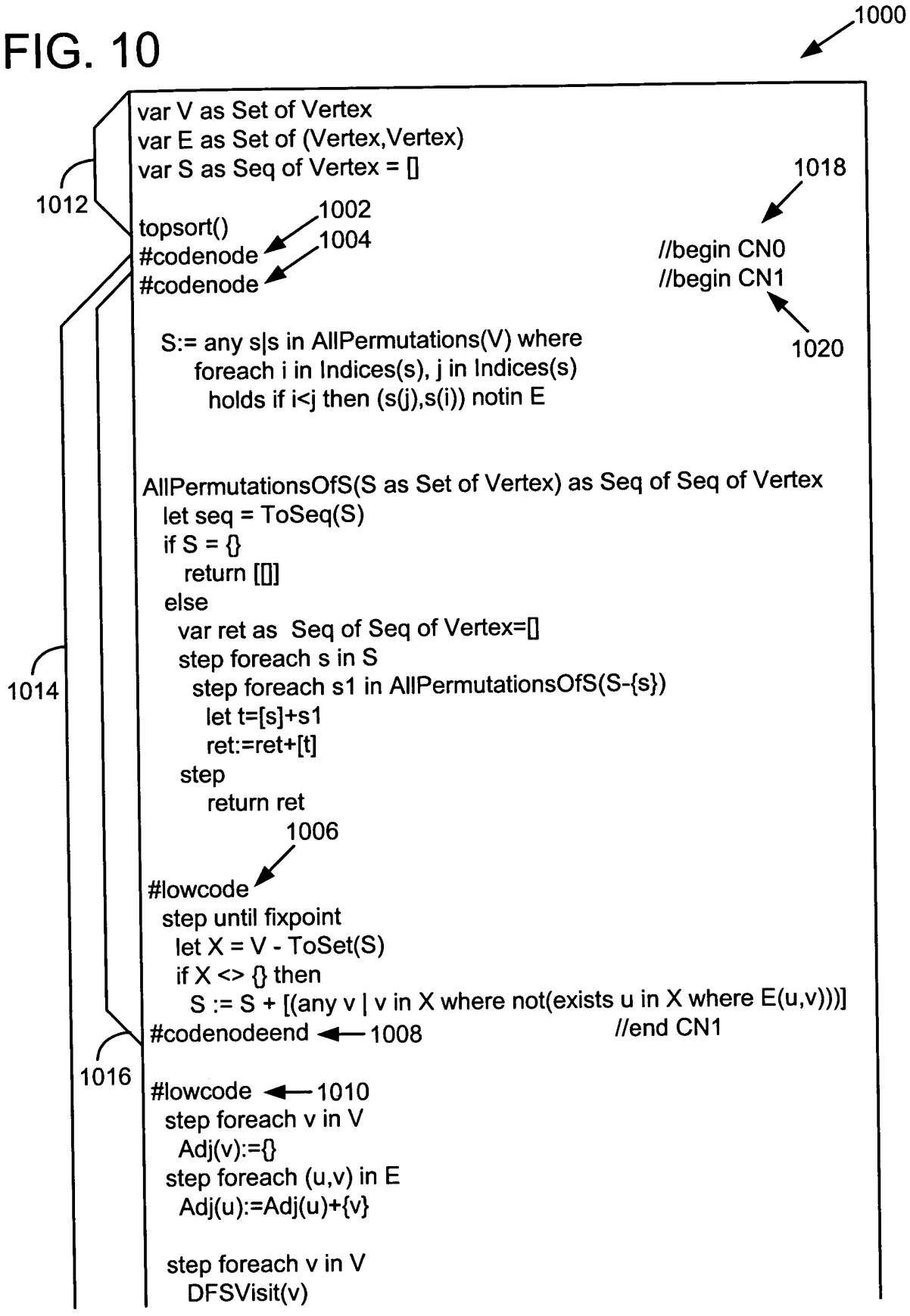

1000
var V as Set of Vertex
var E as Set of (Vertex,Vertex)
var S as Seq of Vertex = []
1012
1018
1002
topsort()
1004
#codenode
#codenode
//begin CN0
//begin CN1
1020
S:= any s|s in AllPermutations(V) where
foreach i in Indices(s), j in Indices(s)
holds if i<j then (s(j),s(i)) notin E
AllPermutationsOfS(S as Set of Vertex) as Seq of Seq of Vertex
let seq = ToSeq(S)
if S = {}
return [[]]
else
var ret as Seq of Seq of Vertex=[]
1014
step foreach s in S
step foreach s1 in AllPermutationsOfS(S-{s})
let t=[s]+s1
ret:=ret+[t]
step
return ret
1006
#lowcode
step until fixpoint
let X = V - ToSet(S)
if X <> {} then
S := S + [(any v | v in X where not(exists u in X where E(u,v)))]
#codenodeend
1008
//end CN1
1016
#lowcode
1010
step foreach v in V
Adj(v):={}
step foreach (u,v) in E
Adj(u):=Adj(u)+{v}
step foreach v in V
DFSVisit(v)

FIG. 11

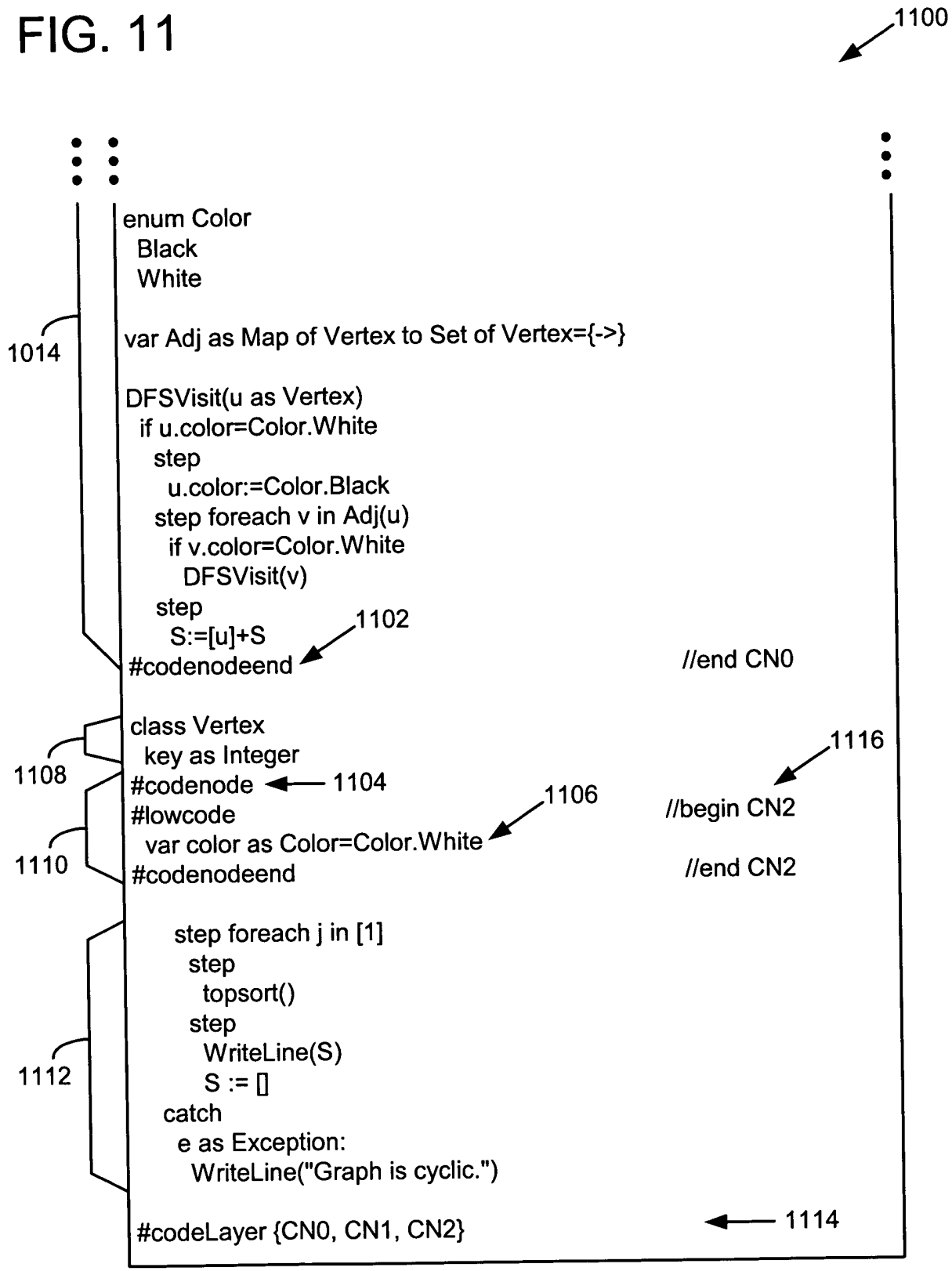

1100
enum Color
Black
White
1014
var Adj as Map of Vertex to Set of Vertex={->}
DFSVisit(u as Vertex)
if u.color=Color.White
step
u.color:=Color.Black
step foreach v in Adj(u)
if v.color=Color.White
DFSVisit(v)
step
S:=[u]+S
1102
#codenodeend
//end CN0
class Vertex
key as Integer
1108
#codenode
1104
1116
#lowcode
1106
//begin CN2
var color as Color=Color.White
1110
#codenodeend
//end CN2
step foreach j in [1]
step
topsort()
step
WriteLine(S)
1112
S := []
catch
e as Exception:
WriteLine("Graph is cyclic.")
#codeLayer {CN0, CN1, CN2}
1114

FIG. 12

1200 →

```
var V as Set of Vertex
var E as Set of (Vertex,Vertex)        } 1202
var S as Seq of Vertex = []

topsort()
  S:= any s|s in AllPermutations(V) where
      foreach i in Indices(s), j in Indices(s)
        holds if i<j then (s(j),s(i)) notin E


AllPermutationsOfS(S as Set of Vertex) as Seq of Seq of Vertex
  let seq = ToSeq(S)
  if S = {}
    return [[]]
  else
    var ret as  Seq of Seq of Vertex=[]
    step foreach s in S
      step foreach s1 in AllPermutationsOfS(S-{s})
        let t=[s]+s1
        ret:=ret+[t]
    step
      return ret


class Vertex                           } 1204
 key as Integer

step foreach j in [1]                  }
      step                             }
        topsort()                      }
      step                             }
        WriteLine(S)                   } 1206
        S := []                        }
   catch                               }
    e as Exception:                    }
      WriteLine("Graph is cyclic.")    }
```

FIG. 13

1300 →

```
var V as Set of Vertex
var E as Set of (Vertex,Vertex)
var S as Seq of Vertex = []

topsort()

  step until fixpoint
    let X = V - ToSet(S)
    if X <> {} then
      S := S + [(any v | v in X where not(exists u in X where E(u,v)))]

class Vertex
  key as Integer

step foreach j in [1]
        step
          topsort()
        step
          WriteLine(S)
          S := []
    catch
      e as Exception:
        WriteLine("Graph is cyclic.")
```

FIG. 14

1400

```
var V as Set of Vertex
var E as Set of (Vertex,Vertex)
var S as Seq of Vertex = []

topsort()
  step foreach v in V
    Adj(v):={}
  step foreach (u,v) in E
    Adj(u):=Adj(u)+{v}

  step foreach v in V
     DFSVisit(v)

enum Color
  Black
  White

var Adj as Map of Vertex to Set of Vertex={->}

DFSVisit(u as Vertex)
  if u.color=Color.White
    step
      u.color:=Color.Black
    step foreach v in Adj(u)
      if v.color=Color.White
        DFSVisit(v)
    step
      S:=[u]+S

class Vertex
  key as Integer
  var color as Color=Color.White

step foreach j in [1]
       step
         topsort()
       step
         WriteLine(S)
         S := []
    catch
      e as Exception:
        WriteLine("Graph is cyclic.")
```

FIG. 15

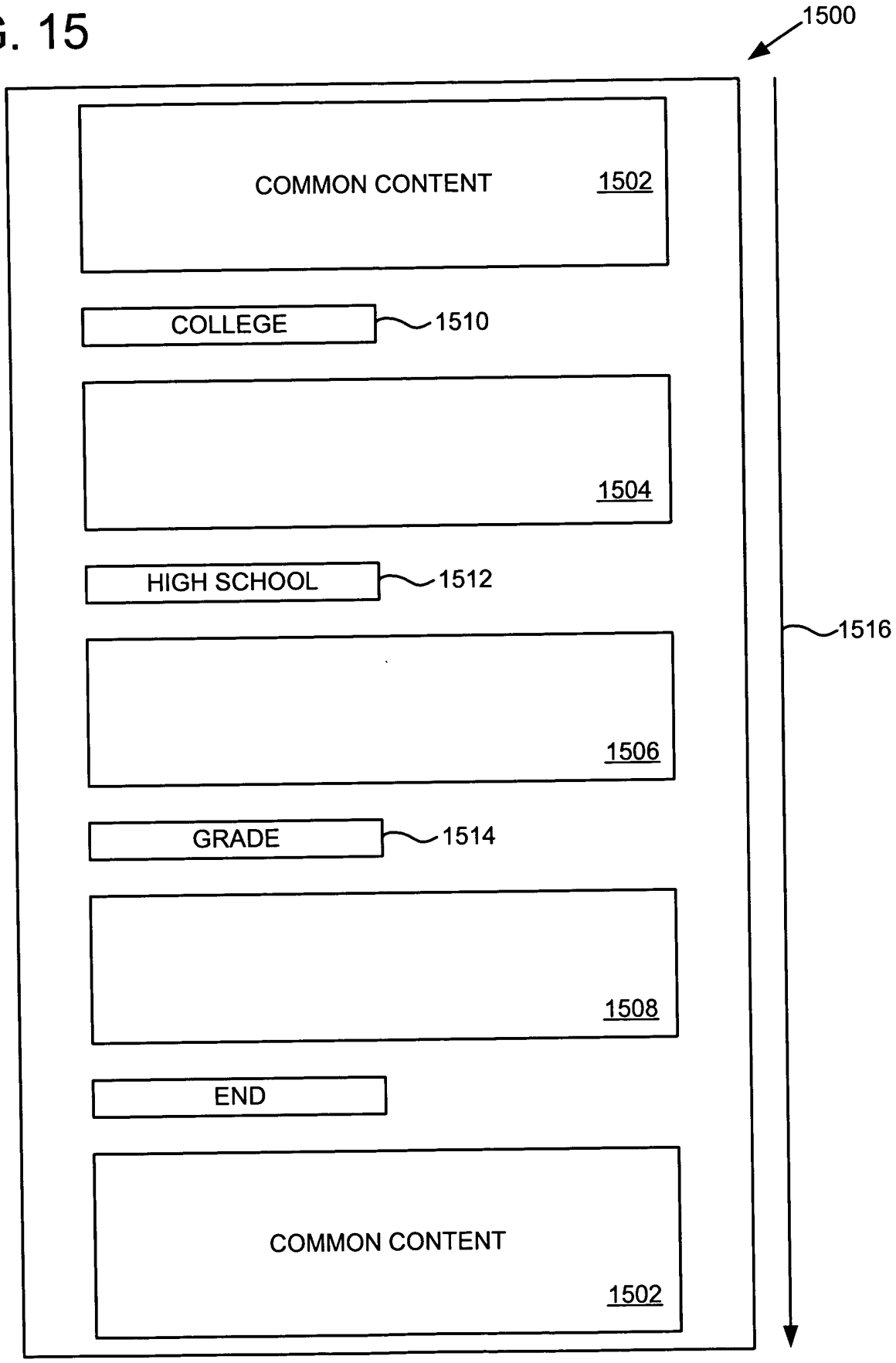
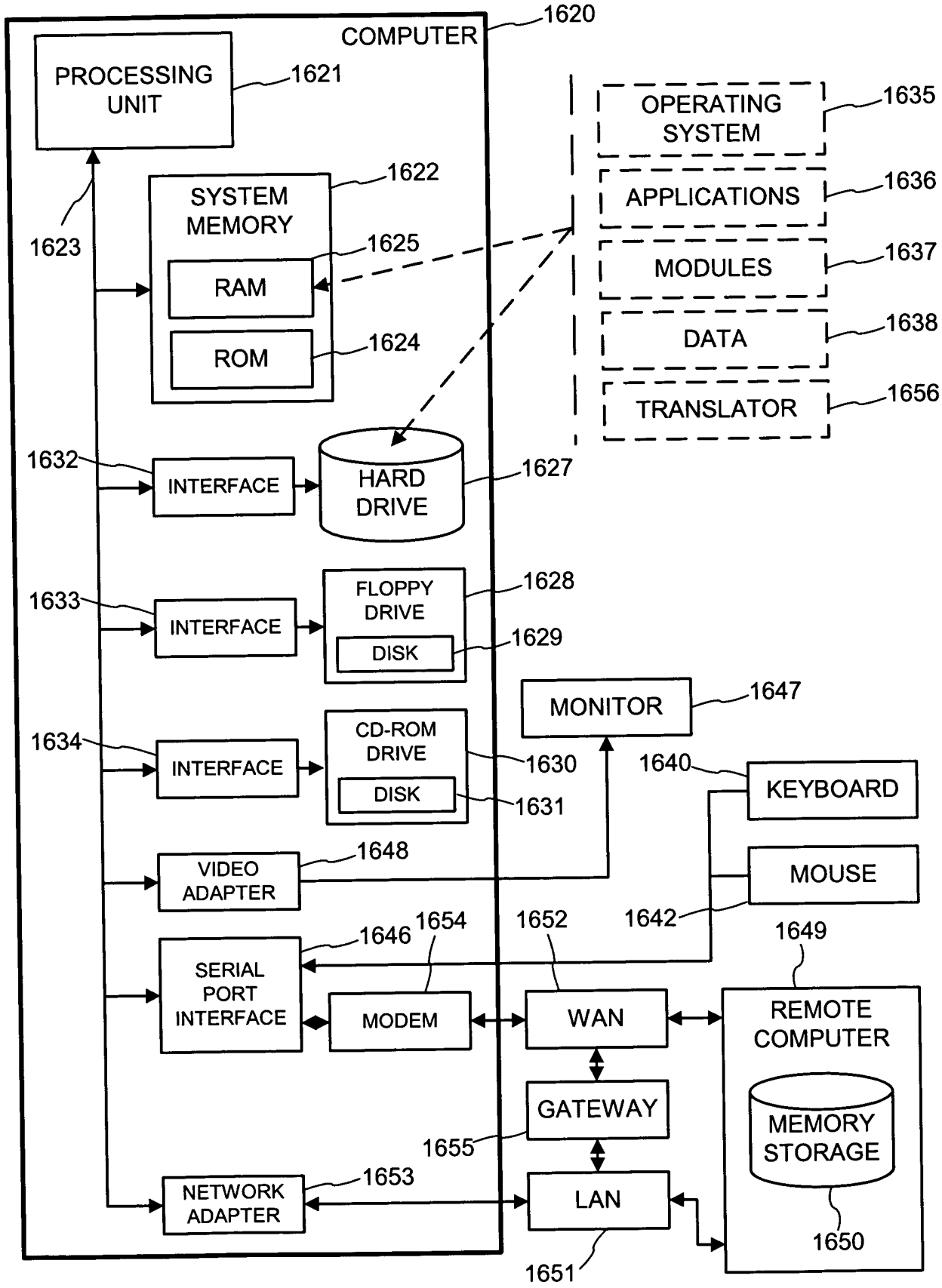

1500
COMMON CONTENT
1502
COLLEGE
1510
1504
HIGH SCHOOL
1512
1516
1506
GRADE
1514
1508
END
COMMON CONTENT
1502

FIG. 16


COMPUTER
1620
PROCESSING UNIT
1621
SYSTEM MEMORY
1622
1623
RAM
1625
ROM
1624
OPERATING SYSTEM
1635
APPLICATIONS
1636
MODULES
1637
DATA
1638
TRANSLATOR
1656
1632
INTERFACE
HARD DRIVE
1627
1633
INTERFACE
FLOPPY DRIVE
1628
DISK
1629
1634
INTERFACE
CD-ROM DRIVE
1630
DISK
1631
MONITOR
1647
1640
KEYBOARD
MOUSE
1642
VIDEO ADAPTER
1648
SERIAL PORT INTERFACE
1646
1654
1652
1649
MODEM
WAN
REMOTE COMPUTER
GATEWAY
1655
MEMORY STORAGE
NETWORK ADAPTER
1653
LAN
1650
1651